MODCOMP
MODULAR COMPUTER SYSTEMS, INC

MAX IV and MAX 32 Programmer's Reference Manual

Link Editor (M4EDIT and LNK32)

SOFTWARE

2111804010-902

MANUAL HISTORY

Manual Order Number: 211-804010-G02

Title: MAX IV AND MAX 32 PROGRAMMER'S REFERENCE MANUAL, LINK EDITOR (M4EDIT AND LNK32)

Software Part Numbers: 610550-000G.1 and 612550-E00A.0

Revision Level	Date Issued	Description
---	08/76	Initial issue (A.0).
---	06/78	Release.
---	02/79	Release.
G00	07/81	Release (G.0). Minor documentation corrections.
G01	08/84	Release (G.1). Reformatted manual to publication standards
G02	07/85	Release (G.1 and A.0). MAX 32 Link Edit information was included.

Contents subject to change without notice.

PROPRIETARY INFORMATION

THIS MANUAL CONTAINS CONFIDENTIAL PROPRIETARY INFORMATION PROTECTED BY SOFTWARE PROPERTY RIGHTS AND IS MADE AVAILABLE UPON THE CONDITION THAT THE SOFTWARE CONTAINED HEREIN WILL BE HELD IN ABSOLUTE CONFIDENCE AND MAY NOT BE DISCLOSED IN WHOLE OR IN PART TO OTHERS WITHOUT THE PRIOR WRITTEN PERMISSION OF MODULAR COMPUTER SYSTEMS, INC.

FOR GOVERNMENT USE THE FOLLOWING SHALL APPLY

RESTRICTED RIGHTS LEGEND

USE, DUPLICATION OR DISCLOSURE BY THE GOVERNMENT IS SUBJECT TO RESTRICTIONS AS SET FORTH IN PARAGRAPH (b) (3) (B) OF THE RIGHTS IN TECHNICAL DATA AND COMPUTER SOFTWARE CLAUSE IN DAR-7-104.9(a) (OR SUCH EQUIVALENT FAR, NASPR, AND/OR DODPR CLAUSE AS MAY BE APPLICABLE).

MODULAR COMPUTER SYSTEMS, INC
1650 WEST McNAB ROAD
FORT LAUDERDALE, FL 33309

PREFACE

Audience
This manual is directed to all users of the MAX IV Link Editor (M4EDIT), product number 610550-000, and users of the MAX 32 Link Editor (LNK32), product number 612550-000.

Subject
This manual describes how to use the MAX IV Link Editor. It contains a general description of the MAX IV Link Editor and a full summary of all directives and error messages.

Product Requirements
The MAX IV Link Editor runs under the MAX IV operating system, Revision H.0 and later, and the MAX 32 Link Editor runs under the MAX 32 operating system, Revision A.0 and later.

Related Publications
Refer to the following manuals for additional information. When ordering manuals, use the manual order number listed below. The most current revision level (REV) will be shipped.

Manual Order Number	TITLE
210-804001-REV	MODCOMP ASSEMBLERS Language Reference Manual (LRM)
205-804001-REV	MAX IV/MAX 32 GENERAL OPERATING SYSTEM Concepts and Characteristics Manual (CCM)
213-804001-REV	MAX IV GENERAL OPERATING SYSTEM System Guide Manual (SGM)
213-838001-REV	MAX 32 GENERAL OPERATING SYSTEM System Guide Manual (SGM)
211-804011-REV	MAX IV AND MAX 32 TASK/OVERLAY CATALOGER Programmer's Reference Manual (PRM)

Special Symbols and Notations
A revision bar (|) located in the margin of the page in the text indicates a change to the manual. This change generally represents a technical change to the product due to product revision. A revision bar is also entered in the Table of Contents to flag the general location of changes in the text.

MODCOMP Product Training
MODCOMP's Education Services provides training courses on many hardware and software products at our Training Center in Ft. Lauderdale, Florida. On-site courses at customer sites can also be arranged. For more information about the courses offered by Education Services, contact the MODCOMP Training Registrar.

REVISION SUMMARY

As of the G02 manual revision level, this manual is a shared source of information for both M4EDIT and LNK32. Unless specific differences are called out in the text, M4EDIT and LNK32 will be referred to as the Link Editor or Link Editors.

LNK32 is also used to link edit MAX 32 SYSGENs. LNK32 output is recognized and used by the 32-bit Stand-alone Loader, SAL32. In the current version of LNK32 (A.0), the link is restricted to a maximum of four (4) megawords.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
OVERVIEW OF THE MAX IV/MAX 32 LINK EDITORS

1.1 PURPOSE OF THE MAX IV/MAX 32 LINK EDITORS

The MAX IV Link Editor program name is M4EDIT. The MAX 32 Link Editor program name is LNK32. Both are normally executed as a background overlay.

M4EDIT is used to link-edit programs to be run in a 16-bit environment under MAX IV or MAX 32. LNK32 is used to link-edit programs to be run in a 32-bit environment under MAX 32. Both Link Editors run under both operating systems.

The Link Editors convert incomplete binary object modules into a complete load module in editor format. The Link Editors allows the user to:

- Define a complex overlay structure (M4EDIT only).
- Associate a program's common areas to private or global shared areas.

The Link Editors are capable of building object modules in an editor format from unedited object modules. They can be used to:

- Convert unedited Assembler/Compiler object to an editor format.
- Segment programs into overlay modules (M4EDIT only).
- Allocate labeled common blocks in the address space.
- Allocate labeled common blocks in extended address space (M4EDIT only).
- Assign attributes such as READ-ONLY, RELOCATABLE, OPERAND-MAP, to multiple location counters and common blocks.
- Associate labeled common blocks with global or private shared regions.

The Link Editors read binary object records in MODCOMP's standard object-language format.

The input record size is normally 80, 200, or 256 bytes per record, depending on the input device. The output record size is equal to the record size specified for the device that records are to be written to, but never exceeding 256 bytes per record.

1.2 OPERATING ENVIRONMENT

The Link Editors require MODCOMP CLASSIC hardware with at least one disc drive and MAX IV or MAX 32 Assembler software.

Output of the MAX IV Link Editor (M4EDIT) is in a format that is recognized only by the MAX IV Task/Overlay Cataloger (TOC). Output of the MAX 32 Link Editor (LNK32) is in a format that is recognized only by MAX 32 Task/Overlay Cataloger(TOC32). LNK32 programs

can only be run under the MAX 32 Operating System. LNK32 is also used to link edit MAX 32 SYSGENS. LNK32 output is also recognized and used by the 32-bit Stand-Alone Loader (SAL 32). Refer to the MAX IV/MAX 32 TASK/OVERLAY CATALOGER, Programmer's Reference Manual for more information.

1.3 LOGICAL FILES USED BY LINK EDITORS

The Link Editors use the following files:

Command Input (CI)	The directives of the Link Editors are read from the CI file when the AO option is reset.
Alternate Input (AI)	The directives of the Link Editors are read from the AI file when the AO option is set.
Input file for main or primary object programs (BI)	This file is specified by the user at editing time through the EDIT Directive. Main object programs are read from the specified BI file.
Library Files	Any file can be used as a library file by naming it in the LIBRARIES Directive. If a LIBRARIES Directive is not specified, the libraries are defaulted to UL (User Subroutine Library) and LB (System Subroutine Library) and are searched in that order. Sequential library files are searched until an end-of-file is encountered or until there are no undefined externals (PASS 1 is over). If at least one module is loaded during the search, the library file is rewound and searched again. The number of searches can be limited by associating a count with a specific file using the LIBRARIES Directive. The count then puts an upper limit on the number of searches of the file. Directoried libraries are searched by reading the directory and searching object modules in the order listed in the directory for missing externals. The message "REWIND FILE - filename" is output on CO before each file search if a library file is on paper tape. The Link Editors use HOLD allowing the user to position the tape. The Operator Communication Directive, /RESUME, is used to resume execution.
Scratch (SC)	All modules loaded during PASS 1 are saved on the SC file and used as the input file during PASS 2.
Listing Output (LO)	Directives and the map listing of the symbol table are written to the LO file.

Binary Output (BO)	The object code for the resulting load module is written on the BO file.
Console Output (CO)	Messages to the operator's attention are written to the CO file.
Scratch File A (SCA)	If the SEGMENT Directive is used and the BI file is not random access, SCA is used if the editor is in the MULTIPLE mode.
Scratch File B (SCB)	If the Shared Load Module (SLM) Directive is used SCB is used as a scratch file.

Figure 1-1 (below, is a graphic description of the files described above:

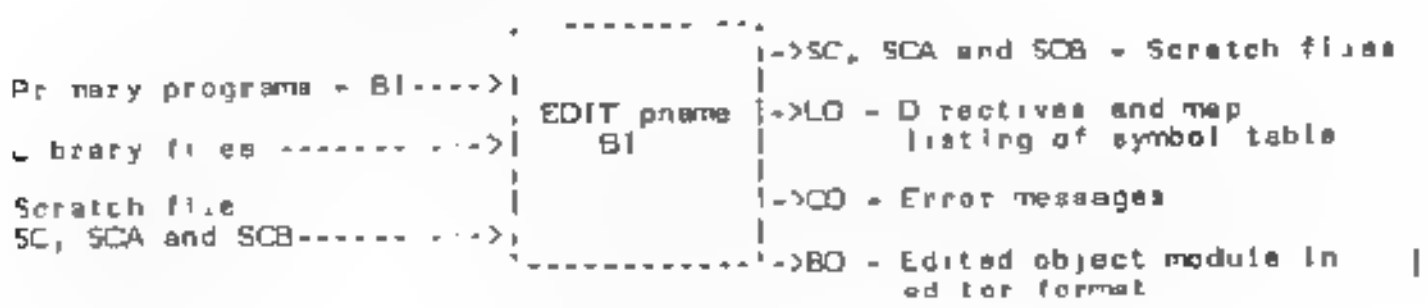


Figure 1-1. File Description

EXAMPLES:

Job streams to compile, link-edit and catalog a FORTRAN program follows (MAX IV on the left, MAX 32 on the right):

MAX IV	MAX 32
$ASS S1=USL S0=SCB LO=LO	$ASS S1=USL B0=SCA
$REW S0	$POS PROGXX
$POS PROGXX	$EXE FTN32
$EXE FR5	$WEOF B0
$WEO S0	$ASS B1=SCA B0=SCB
$ASS S1=S0, B0=SCA	$REW B1
$REW S1 B0	$EXE LNK32
$EXE M5A,,NOLO	EDIT MAIN B1
$WEOF B0	WEO B0
$ASS B1=SCA, B0=SCB	$EXIT
$REW B1 B0	$ASS B1=SCB
$EXE M4EDIT	$REW B1
EDIT MAIN,B1	$EXE TOC32
WEOF B0	OVE filename
EXIT	CAT
$ASS B1=SCB	EXIT
$REWIND B1	
$EXE TOC	
OVE filename	
CAT	
EXIT	

1.4 SUMMARY OF USE AND FUNCTION

An Assembly Language program referencing names that are all defined within itself is a complete program. When a complete program is assembled, its binary object output (generated by an assembler) is referred to as the program's relocatable object module. An Assembly Language program referencing names not defined within itself (Example: external labels) is an incomplete program. When an incomplete program is assembled, its binary object output (generated by an assembler) does not constitute a complete relocatable object module because the memory addresses for the external names are not satisfied.

An incomplete program module is link-edited with other object modules to satisfy the missing address. These other object modules contain internal name definitions identical to the names referenced as Externals in the module. If subsequent modules have their own external label references, externals can be in any of the previous modules. The first object program encountered by the Link Editors is referred to as the primary program module. The primary program module is read from a user specified file. In trying to satisfy missing external names, object programs are read from several library files (Example: UL and/or LB). Libraries are searched in the order specified by the LIB Directive, or the default order of UL then LB if the LIB Directive is absent.

The link-editing process requires the following passes over the input modules.

- PASS ONE

  The primary object module is read and a memory resident symbol table is constructed of all internal definitions, external references, and common allocation. All external references are then resolved from either the specified input file, the optional user library file (UL), the system library file (LB), or from user specified libraries. The validity of all data is checked during this pass.

  If there are any unsatisfied external references, a search of the libraries is performed. The internals of each module in the libraries are checked against the set of unsatisfied externals until all the necessary modules are loaded. When segmentation is used (refer to the SEGMENT Directive) library searches are performed at each change of level in the structure if necessary.

- SECOND PASS

  During the second pass all references to external names are satisfied from the symbol table. A self-contained load module is written on the Binary Output (BO) File.

1.5 SEGMENTS (M4EDIT only)

Segmentation permits a program requiring large memory allocation to overlay itself, therefore requiring less memory. Flexibility in generating overlay modules is made possible by the use of the SEGMENT Directive.

1.6 COMMON BLOCKS

FORTRAN or Assembly Language programs can reference global shared areas under the operating system. Any labeled common block in the user's program can be associated with a system global shared area at link-edit time by means of the COMMON Directive. Labeled common blocks can also be associated with private shared areas and shared load modules.

1.7 EXTENDED MEMORY (M4EDIT only)

Common blocks defined as being in extended memory are allocated space by M4EDIT. All references to these common blocks are resolved based on that allocation.

1.8 OPTIONS

The Link Editors provide the following options:

- LO - If the LO option is reset, directives are not listed on the LO file.
- AO - If the AO option is set, EDIT Directives are read from the AI file (instead of the CI file).
- MAP - If the MAP option is reset, the Map listing of the symbol table is bypassed.

$HO - If the HOLD option is set, the Link Editors enter the HOLD state on each error. When CI is assigned to a terminal device, no HOLD occurs. HOLD is the default value.

$SC - (LNK32 only) If the SCROLL option is set when the CI is assigned to a terminal device, the Link Editor will output more than one page at a time to CO. Default is NOSCROLL.

1.9 SEGMENTATION (MAEDIT only)

Segmentation is required under MAX IV if:

- The program's size exceeds available memory, or
- The program's size exceeds the addressing space available.

To develop programs that can run on different configurations, develop the program modularly and reorganize its use of memory at a later time. This reorganization typically requires the sharing of available memory by several of the program's modules. Figure 1-2 exhibits a program next to the available addressing space that is too small to contain the program.

Figure 1-2. Program and Addressing Space

The main program references SUB1, SUB2, SUB3, and SUB4, however, none of these reference each other. To fit the program into the address space, SUB1, SUB2, SUB3, and SUB4 can be constructed to share the space currently occupied by SUB1. The program layout in memory can then be represented as in Figure 1-3.

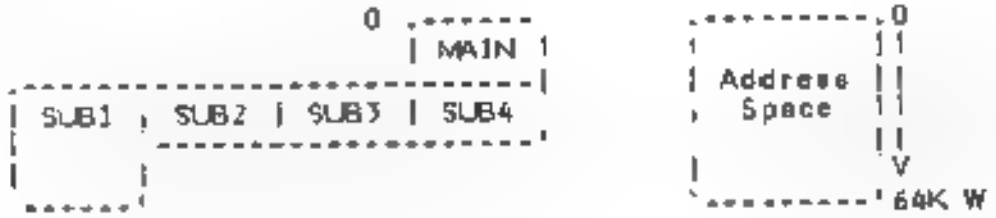


Figure 1-3. Program Segmentation

The reorganization of a program in this manner is called segmentation. Each module is called a segment. If SUB1, SUB2, SUB3, and SUB4 reference subprograms in a similar manner, further size reduction can be made as shown in Figure 1-4.

Figure 1-4. Host-Guest Relationships

The terms host and guest are used to express the relationships of the segments. The main program segment in Figure 1-4 is a host in that it "invites" the execution of and provides common data areas and subroutine services to SUB1, SUB2, SUB3, and SUB4. SUB1, SUB2, SUB3, and SUB4 are guests of the main program. While SUB1 is a guest of the main program, it is the host to SUB1.1, SUB1.2, and SUB1.3.

In the tree-like organization of the segmentation process, there is one host that cannot be considered a guest of another host. This host corresponds to the root of the tree and is called the master host.

The host segment makes a reference to one or more guest segments. Guest segments can occupy the same addressing space at different times. The host must occupy addressing space during the execution of a guest so that references can be made to the host from the guest.

A guest can exist in three states:

NOT LOADED
LOADED
ACTIVE

This is illustrated in Figure 1-5.

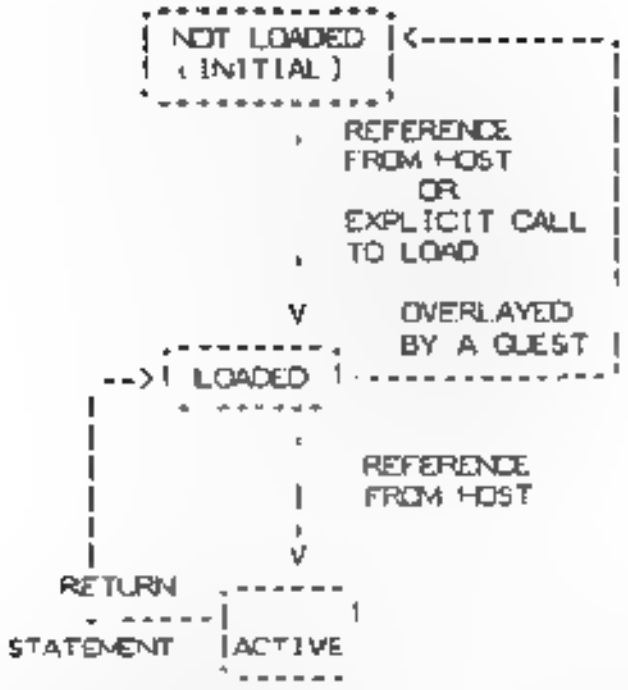


Figure 1-5. Guest Processing States

NOT LOADED is the initial state that describes the existence of a segment when it is not in the addressing space of a program. The execution of a guest reference or an explicit call to the segment manager to load the segment causes the following:

The runtime segment manager (a program that loads programs when they are called) loads the program into the program addressing space.

The guest is placed in the LOADED state.

The runtime segment manager transfers execution control to the guest and places it into the ACTIVE state.

- An exit from the guest places the guest in the LOADED state.

References to a guest in the LOADED state do not cause a reload of the guest.

A guest can become ACTIVE as the result of host references during the guest's (and the required hosts) existence in the programs addressing space. The guest is returned to the "NOT LOADED" state when overlayed or when one of its required hosts is overlayed.

The SEGMENT Directive is used to achieve this segmentation at link-edit time.

The designer of a segmented program must be aware of the following rules:

A guest's host must be loaded before the guest can execute.

- Guests of a common host cannot reference each other.

CHAPTER 2
SUMMARY OF DIRECTIVES

Summaries of the directives processed by the Link Editors follow in alphabetical order. A detailed description of each directive and its syntax is given in Chapter 3.

ACTION	Write a message to the operator on the CO file, then HOLD.
ALLOCATE	Provides memory for segments (M4EDIT only).
ASSIGN	Causes assignments of logical files to logical devices or other logical files.
ATTRIBUTE	Allows user control over two-map tasks, program relocation, access rights, and shared regions.
AVFILE	Causes a device to go forward specified number of file marks.
AVRECORD	Causes a device to go forward a specified number of physical records.
BIAS	Positions a program in virtual space.
BKFILE	Causes a device to go backwards a specified number of file marks.
BKRECORD	Causes a device to go backward a specified number of physical records.
COMMON	Positions labeled common blocks in virtual storage and associates user's labeled common blocks with established shared area. Refer to the note at the end of this summary.
DISPLAY/NODISPLAY	DISPLAY echos I O output to CO. NODISPLAY does not echo output. (LNK32 only).
DEFX	Allows external reference definitions at execution of Link Edit.
EDIT	Search, load and link-edit a specific program name.
EXCLUDE	Specifies internal definitions that are to be excluded from indicated module.
EXIT	Causes the operating system to load and transfer to Job Control.
INCLUDE	Causes unreferenced subprograms to be incorporated in specific modules.
INITIALIZE	Removes all associations and symbols, then returns to the initialized state.
LIBRARIES	Sets up library files for subprogram searches. Refer to the note at the end of this summary.

MULTIPLE	Edits all modules or main program depending on usage of SEGMENT Directive.
NOTE	Causes a message for the operator's attention to be written to the logical file CO while a job stream is executing.
NOXEQUATE	Remove all external name associations previously entered by the XEQUATE Directive (M4EDIT only).
ONEMAP	Causes operands and instructions to share the same map addressing space (M4EDIT only).
PAUSE	Causes a message for the operator's attention to be written to the logical file CO, and then to HOLD.
PRIMARY	Causes the Link Editors to read only one object module from the BI file in subsequent editing operations.
REWIND	Positions device to beginning of media.
SCROLL/NOSCROLL	SCROLL causes continuous output to CO if CI is a terminal. NOSCROLL causes paging to CO if CI is a terminal (LNK32 only).
SEGMENT	Defines the desired organization of a segmented program. Refer to the note at the end of this summary (M4EDIT only).
SLM	Separates all counters with attributes in attribute list to form shared load modules.
TWOMAP	Causes the module to be formed so that different operand and instruction maps are used. ONEMAP is the default (M4EDIT only).
WEOF	Writes a file mark on each logical file name specified.
XCO	Defines the name of all extended common blocks resolved by the current edit (M4EDIT only).
XEQUATE	Associates two parameters in the symbol table so that external references made to Parameter 1 can be satisfied by internal references of Parameter 2.

NOTE: The order of parameters is important in the following three directives:

- COMMON
- LIBRARIES
- SEGMENT

Repeat the directive's keyword to continue these directives on the following line. The editor treats the new line as a continuation of the line above it, instead of a new directive as is the case for all other directives.

CHAPTER 3
LINK EDITOR DIRECTIVES

The Link Editors read directives (command sentences) and performs them. Some directives perform utility functions, some set the operation mode of subsequent directives, and others perform editing functions.

The format used in the presentation of this information is as follows:

KEYWORD

Brief Summary

Description in paragraph form.

SYNTAX

KEYWORD parameter1, parameter2

parameter1 - (Description in paragraph form)

parameter2 - (Description in paragraph form)

EXAMPLES

Symbols used:

xxx	Lower-case letters represent a variable, the value of which is supplied by the user.
XXX	Upper-case keywords are to be entered as shown.
[ ]	Brackets indicate either insignificant characters or an optional parameter.
{ }	Braces indicate a set of options from which the user is required to choose at least one.

The following sections present each of the Link Editors directives in alphabetical order.

ACTION

Writes A Message To Operator, Then Holds

The ACTION Directive writes a message to the operator on the CO file. After the message is written, the task enters the HOLD state.

SYNTAX

ACT[ION] [message]

[message] - Parameter 1 (optional) is a message directed to the operator's attention.

EXAMPLE

ACT MOUNT MAGTAPE ON MT2

Provides Memory For Segments

The ALLOCATE Directive has an effect only if segmentation is used (refer to the SEGMENT Directive).

SYNTAX

ALL[OCATE] {ALL }
{ROOT}
{SEG }

{ROOT} - Parameter 1 (at least one is required) specifies the segments that
{ALL } memory is provided for.
{SEG}

ROOT (default) specifies memory allocation for the main (or root) segment only when it is loaded. Segments allocate memory as needed only when loaded.

ALL specifies the memory necessary to hold all the segments allocated when the root segment is loaded.

SEG allocates memory only for the segment being loaded. All of the remaining memory up to the high address of the edit is deallocated. Any SPACE given to the module by Task/Overlay Cataloger (TOC) is not effected.

EXAMPLES

```
ALLOCATE   ALL
ALL ROOT
```

Delegates Files To Devices Or Other Files

The ASSIGN Directive assigns logical files to logical devices or to other logical files. A logical file assigned to itself is assigned to its default assignment in the system.

SYNTAX

ASS[IGN] file devicename
filename

file - Parameter 1 (required) is a file name.

devicename - Parameter 2 (required) is the name of a device filename or file.

EXAMPLES

```
ASS BI TY
ASSIGN,LO=8 ,BO=DMA
```

Allows User Control Over Access, And Regions

The ATTRIBUTE Directive allows the user to exercise some control over how a program is treated when loaded and when executing. The main areas controlled by this directive are:

- Two-map tasks (M4EDIT only)
- Relocation of programs
- Access rights
- Shared regions

The CLASSIC Macro Assembler statement ATR can be used to pass 32,767 sets of attributes from symbolically assembled code to the Link Editors. The user of the Assembler and the Link Editors must establish a relationship between the assembler produced attribute code and actual execution time attributes. There are four types of attributes:

- Address Relocation
- Map Image
- Access Right
- Sharing

SYNTAX

ATT[RIBUTE] {number|uname|gname},att...,[RKEY=xxx,][WKEY=yyy,][LFN=zzz]

{number|uname|gname} - Parameter 1 (at least one is required) represents one of the following:

number specifies the attribute set number generated by the assembler.

uname specifies the name of a labeled common block.

gname specifies the name of a shared area (global or private).

att... - Parameter 2 (at least one is required) represents one of the attributes defined as follows. (Only one attribute can be chosen from each type.) Refer to Table 3-1 or 3-2 for more information on attributes.

ADDRESS TYPE

ABS - All programming elements associated with the attribute set are loaded into absolute virtual addressing space.

REL - All programming elements associated with the attribute set can be relocated in virtual addressing space.

MAP TYPE (M4EDIT only)

IMAP — All programming elements associated with the attribute set are accessible in the instruction map assigned to the task.

OMAP — All programming elements associated with the attribute set are accessible in the operand map assigned to the task.

ACCESS RIGHTS TYPE

REA(D) ONLY
Access rights in the map image are set to Read Access Only.

EXE(CUTE)
Set to READ and EXECUTE ACCESS.

NOR(ESTRICTIONS
Set to No Restrictions.

WRI(TE) (LNK32 only)
Set to WRITE ACCESS.

SHARING TYPE

GLO(BAL)
This attribute is associated with a shared global region defined in a COMMON Directive that is to be inserted into the task's virtual space at load time. The global shared region must have been defined at system generation time. An INSERT GLOBAL SHARED REGION is generated.

IPR(IVATE) (Insert Private)
This attribute is associated with a private shared region defined in the COMMON Directive that is to be inserted into the task's virtual space at load time. For the insert to work the region must have been created by another task. An INSERT IPRIVATE SHARED REGION is generated.

CPR(IVATE (Create Private)
This attribute is associated with a private shared region defined in the COMMON Directive that is to be created at load time. A CREATE PRIVATE SHARED REGION is generated.

SLM (Insert Shared Load Module)
This attribute is associated with shared load modules defined by the SLM and COMMON Directives. An Insert Shared Load Module is generated.

RKEY=xxx - Parameter 3 (optional) represents a READ key associated with a shared area.

WKEY=yyy - Parameter 4 (optional) represents a WRITE key associated

LFN=zzz - Parameter 5 (optional) represents the logical file name of the Shared Module to be inserted. The variable must consist of three CAN-code characters. If Parameter 5 is missing, the module is loaded from the last LFN used to load programs.

EXAMPLES

```
ATTRIBUTE 11,ABS,IMAP,EXECUTE
ATT 99,REL,OMAR,NORESTRICT
ATT COMMON,REL, OMAP,NOR, IPR, RKEY=OLK,WKEY=KLD
```

Table 3-1 shows the illegal combinations of attributes that can be assigned to an attribute set number. To determine whether an attribute can be added to the list, take the logical sum of the 1's of the existing attributes from the row. If an 1 appears in the logical sum corresponding to the desired attribute in the column, the attribute cannot be added.

	ABS	REL	IMAP	OMAP	READ ONLY	EXECUTE	NORESTRICT	CREATE PR, VATE	INSERT PRIVATE	GLOBAL	SHARED LOAD MODULE
ABS	X	I									
REL	I	X									
IMAP			X	I							
OMAP			I	X							
READONLY					X	I	I				
EXECUTE					I	X	I				
NORESTRICTIONS					I	I	X				
CREATE PRIVATE								X	I	I	I
INSERT PRIVATE								I	X	I	I
GLOBAL								I	I	X	I
SHARED LOAD MODULE								I	I	I	X

Table 3-1. Illegal M4EDIT Attribute Combinations

	ABS	REL	READONLY	EXECUTE	NORESTRICT	WRITE	CREATE PRIVATE	INSERT PRIVATE	GLOBAL	SHARED LOAD MODULE
ABS	X	I					I	I		
REL	I	X					I			
READONLY			X	I	I	I				
EXECUTE			I	X	I	I				
NORESTRICTIONS			I	I	X	I				
WRITE			I	I	I	X				
CREATE PRIVATE							X	I		I
INSERT PRIVATE							I	X	I	I
GLOBAL							I	I	X	I
SHARED LOAD MODULE							I	I	I	X

Table 3-2. Illegal LINK32 Attribute Combinations

DEFAULT AND RESERVED ATTRIBUTES

Counters 1-12 not explicitly assigned an attribute set number in the source code (by using the CTR or ATR Directive) are defaulted to the attribute set number equal to the counter number. For example, counter 1 is defaulted to attribute set 1, counter 6 is defaulted to attribute set 6, and so forth.

Attribute set numbers 1-12 are defaulted to the set of attributes indicated below. Override these assignments by using the ATTRIBUTE Directive.

MAX IV

ATTRIBUTE SET #	ATTRIBUTES
1	IMAP,REL,NORESTRICTIONS
2	IMAP,REL,EXECUTE
3	OMAP,REL,NORESTRICTIONS
4	OMAP,REL,READONLY
5	IMAP,REL,READONLY
6	IMAP,ABS,NORESTRICTIONS
7	IMAP,ABS,EXECUTE
8	OMAP,ABS,NORESTRICTIONS
9	OMAP,ABS,READONLY
0	IMAP,ABS,READONLY
11	OMAP,REL,EXE
12	OMAP,ABS,EXE

MAX 32

ATTRIBUTE SET #	ATTRIBUTES
1	REL,NORESTRICTIONS
2	REL,EXECUTE
3	REL,NORESTRICTIONS
4	REL,READONLY
5	REL,READONLY
6	ABS,NORESTRICTIONS
7	ABS,EXECUTE
8	ABS,NORESTRICTIONS
9	ABS,READONLY
0	ABS,READONLY
11	REL,EXE
12	ABS,EXE

Counter 0 is a special case and is defaulted to attribute set 6 if not explicitly assigned an attribute set number. In any case, counter 0 always has an attribute of absolute.

Counters 13-31 default to attribute set 1 if not explicitly assigned an attribute set number.

Any attribute set number can be assigned a set of attributes at link-edit time using the ATTRIBUTE Directive. If no default exists, they are assigned to attribute set 1.

The default attributes associated with a shared region or a local common block are:

OMAP,RELOCATABLE,NORESTRICTIONS

Any shared area that is not explicitly assigned a sharing attribute is assumed to be GLOBAL.

When labeled common blocks are assigned to a shared region, (by the COMMON Directive), they take on the attributes of the shared region. If any attributes have been explicitly assigned to the labeled block, they are ignored.

Advances File Specified File Marks

The AVFILE Direct ve advances a device medium in the forward direction unt l the specified number of f,.e marks have been read.

SYNTAX

AVF[ILE] f le [n]

file - Parameter 1 (required) specifies a file name.

n - Parameter 2 (optional) specifies the number of f,.e marks to be advanced. (Default value = 1)

EXAMPLES

AVF ABC,2
F,.e ABC is advanced two file marks.

AVF DEF
F,.e DEF is advanced one f .e mark.

AVRECORD

Advances File Specified Records

The AVRECORD Directive moves a device media in the forward direction until the specified number of physical records have been skipped.

SYNTAX

AVR[ECORD] file [n]

file - Parameter 1 (required) specifies a file name.

n Parameter 2 (optional) the number of records to be advanced (skipped). (Default = 1)

EXAMPLES

AVR GR5 2
File GR5 is advanced two records.

AVR TUV
File TUV is advanced one record.

Position Program In Virtual Space

The BIAS Directive positions a user's program in virtual space. If ONEMAP is used, only Parameter 1 is used for the loading start address.

SYNTAX

BIA[S] [address] [,oaddress]

[address] - Parameter 1 (optional) specifies the address to start loading relocatable code associated with the instruction map.

[oaddress] - Parameter 2 (optional) specifies the address to start loading relocatable code associated with the operand map (M4EDIT only).

EXAMPLES

```
BIAS #400, #200
BIA #1000
```

Backs Up Device Specified File Marks

The BKFILE Directive moves a device media in the reverse direction until the specified number of file marks have been reached.

SYNTAX

BKF[ILE] file [n]

file - Parameter 1 (required) specifies a file name.

n - Parameter 2 (optional) specifies the number of file marks to be backspaced. (Default value = 1)

EXAMPLES

```
BKF $O
BKFILE,BO,8
```

Backs Up Device Specified Records

The BKRECORD Directive moves a device media in the reverse direction until the specified number of physical records have been skipped.

SYNTAX

BKR[ECORD] file [n]

file - Parameter 1 (required) specifies a file name.

n - Parameter 2 (optional) specifies the number of records to be backspaced. (Default value = 1)

EXAMPLES

```
BKR SO
BKREC,S1,3
```

COMMON

Positions Common Blocks And Makes Associations

The COMMON Directive performs the following functions.

- Positions labeled common blocks in virtual space.
- Associates labeled common blocks with shared areas established under the MAX IV or MAX 32 Operating Systems or with established shared areas of the task.

SYNTAX

COM[MON] ,sname, ssize, [saddress] /lname, lsize, [laddress]
[offset]

Parameters 1 through 3 represent the shared-area parameters.

NOTE: Null represents the absence of a parameter between delimiters.

sname — Parameter 1 (required) represents the shared area name.

ssize — Parameter 2 (required) represents the number of words in the shared area.

saddress — Parameter 3 (optional) represents the location in vertical space to position the shared area.

Parameters 4 through 6 represent the labeled block parameters.

name — Parameter 4 (required) represents the labeled common block name.

lsize — Parameter 5 (required) represents the number of words in the labeled common block.

[laddress] [offset] — Parameter 6 (optional) represents one of the following:

laddress is the location that the non-shared labeled block in the virtual space is to be positioned to.

offset is the starting location of the labeled block relative to the start of the shared area.

EXAMPLES

```
COMMON /GBLOCK,1024/A,B/SHARED,#200/C,D,50,E
COM  /GLOB,#1000/A,#100,#25,B,#100,#50,/D,E/SYS,256/Z
COM  A, B, C/GCOM, 256/D
```

If a shared area is a global region, the required size parameter must equal the size designated at system generation. If a shared area is a private region to be inserted, the size must equal the size given by the creating task. If a shared area is a private region to be created, the user has control over the size. If the optional address parameter is given for any shared area it must be on a page boundary.

The optional address parameter associated with a labeled block name need not be on a page boundary and is of two types: "laddress" and "offset". Parameter "laddress", that is for a labeled block not associated with a shared area, positions the labeled block in the virtual space. Parameter "offset" is for a labeled block associated with shared areas and is used to position it inside the shared area. The "offset" address is relative to the start of the shared area.

All blocks that are not given specific addresses are allocated space in the order they appear in the COMMON Directive (with respect to instruction and operand map defined by the ATT Directive).

Parameters "address" or "saddress" may be preceded by a '$' indicating that the BIAS associated with the given map be added to the address given. A '$' preceding "offset" is ignored.

A double slash (//) indicates the following labeled blocks are not associated with a shared area.

All programs that have been through a compilation process must have declared when link-editing rather than when cataloging the program with TOC. Tasks that access common must use the PAGESHARE Directive in TOC.

EXAMPLE 1

Assume that a FORTRAN program contains the following COMMON statement:

```
COMMON /COMA/A(20), B(20)/COMB/I(30), J(10)
```

1. Both common areas, COMA and COMB, are to be associated with a global shared area (global common) called GLCOM defined at system generation to be 256 words in size. COMB is to follow COMA. At link-edit time, the appropriate COMMON Directive would be:

```
COMMON /GLCOM,256/COMA,COMB
```

When the task or overlay is loaded, GLCOM is inserted in the user's virtual space at location 0 (or the location specified by the BIAS Directive):

2. Assume that GLCOM is to be inserted at location 512 (#200). In addition, COMA is to be offset #14 locations from the start of GLCOM and COMB is to be offset #C8 locations from the start of GLCOM. This is accomplished by the following COMMON Directive:

```
COMMON /GLCOM,256,#200/COMA,,#14,COMB,,#C8
```

When explicitly assigning addresses in this way, issue a BIAS Directive so that the body of the task or overlay does not coincide with the common block as follows:

```
BIAS #300
```

The user's space now looks like:

Preferably, do not assign addresses explicitly. Instead, let the Link Editors assign locations. The use of parameters "saddress" and "laddress" is discouraged unless absolutely necessary. The parameter "offset" is not a problem.

If GLCOM is not a global common area but is a private shared region to be inserted or created by the task or overlay, the ATTRIBUTE Directive must be used in addition to the COMMON Directive.

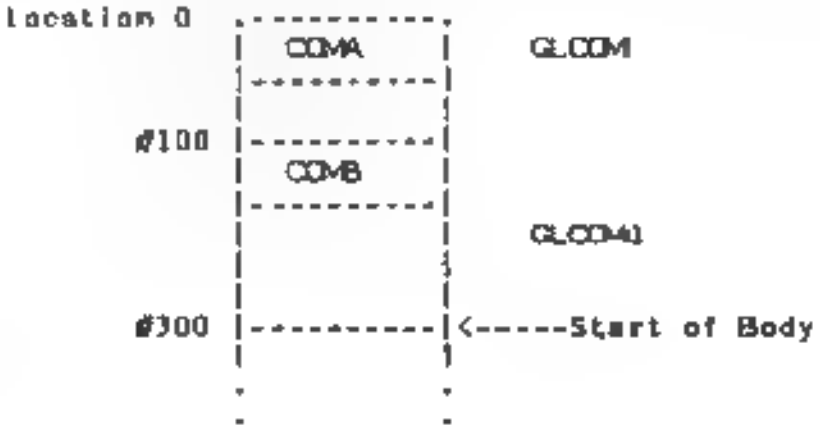
EXAMPLE 2

Assume the following FORTRAN program is the same as in Example 1. To associate common block COMA with global common area GLCOM again, but also associate COMB with a different global common area, GLCOM1, use the following COMMON Directive:

```
COMMON /GLCOM,256/COMA/GLCOM1,512/COMB
```

(assuming GLCOM1 was defined to be 512 words long).

The user's space would now look like this:



EXAMPLE 3

Assume that a user has defined three labeled common blocks in the main program called A1, A2, and A3. For purposes of using overlays, the guerenteed order of these blocks is assigned to memory as A3, followed by A2, followed by A1.

The directive:

```
COMMON A3,A2,A1
```

insures the above requirements.

EXAMPLE 4

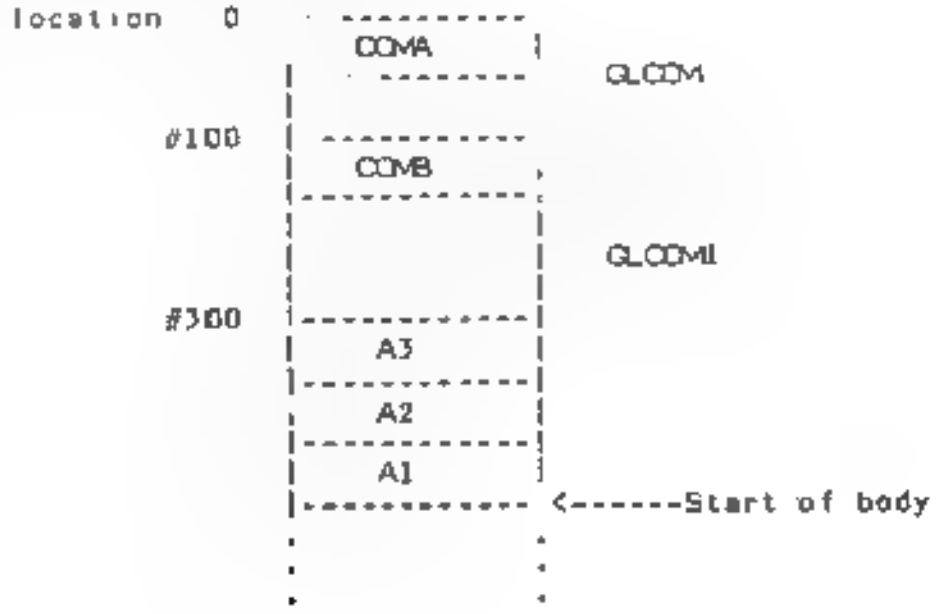
Examples 2 and 3 can be combined into one COMMON Directive as follows:

```
COMMON /GLCOM,256/COMA/GLCOM1,512/COMB//A3,A2,A1
```

The double slash (//) following COMB indicates the following local common blocks are not to be associated with any shared area. An alternate form of the directive would be three consecutive directives (order does make a difference):

```
COMMON /GLCOM,256/COMA
COMMON /GLCOM1,512/COMB
COMMON A3,A2,A1
```

In either case, the layout of the user's space would look like:



The sizes of A3, A2, and A1 are not considered above.

EXAMPLE 5

The following example illustrates the link-edit and TOC commands necessary for successful creation of the following FORTRAN program where the labeled common area TEST is to be an inserted, private shared region.

```
1           IMPLICIT REAL*8 (A-H)
2           IMPLICIT INTEGER*2 (I-N)
3           COMMON /TEST/I(50),A(20)
4  100      CONTINUE
5           A(1)=1.0
6           PAUSE
7           A(1)=2.0
8           PAUSE
9           GOTO 100
10          END
```

Assuming the source has been compiled and assembled and the object is lying on logical file 81, the following directives would have to be typed:

```
$REW 81 80
$EXE M4EDIT
COMMON /TEST,180/TEST,180
ATTR TEST IPR
EDIT MAIN 81


WEO 80
EXI
$EXE TOC
ASS 81 80

PAGESHARE 1
FIL LM
TASK AM.FT4
REW 81
CAT
EXI
```

DEFX

Defines External References

The DEFX (DEFine EXternal) Directive allows the definition of External references at link-editing time by the user.

SYNTAX

DEF[X] name address [,name,address]...

name - Parameter 1 (required) specifies a symbolic name (up to six characters long) that is used in an External definition.

address - Parameter 2 (required) specifies a memory address and can be a decimal, hexadecimal, or relative hexadecimal number.

EXAMPLES

```
DEF M$A,2304
DEFX M4B,#3245
DEFX M$C,$#456
```

Echo/Don't Echo LO To CO

The DISPLAY/NODISPLAY Directive allows the user to display a copy of the LO output on CO. Normally, map output is sent to LO only. Default is NODISPLAY.

SYNTAX

DIS[PLAY]
NOD[ISPLAY]

EXAMPLES

DIS
NOD

Search, Load And Link-Edit Specific Program Name

The EDIT Directive causes the processor to search the file specified in Parameter 2 starting from its current position, for the specified program name.

SYNTAX

EDI[T] [MAIN ,b]
[pname]

MAIN [pname] - Parameter 1 (optional) specifies a program name. (Default = MAIN)

bi - Parameter 2 (optional) specifies a file name. (Default = BI)

EXAMPLES

EDI MAIN,S
EDIT FOX,BI
EDIT,OOG,SI

When Parameter 1 "pname" is found, the program is loaded and link-edited. If MAIN is entered, the first object module that is read from the file specified in Parameter 2 is link-edited, regardless of the program name specified within that module. "MAIN" and "BI" are defaults if only EDIT is entered.

The MULTIPLE and PRIMARY Directives are used to set up the mode of operation for the EDIT Directive. Figure 3-1 shown at the end of Section 3.2 presents a general picture of the EDIT Directive function.

Specifies Internal References To Be Ignored

All of the internal definitions in the subprogram to be excluded must be specified in the directive for the module to be excluded from the resultant editor output.

SYNTAX

EXC[LUDE] pname,I-name[,I-name...]

pname - Parameter 1 (required) represents the name of a program which contains the references to the internals to be excluded.

I-name - Parameter 2 (required) names the internal definitions in the subprograms to be excluded from the indicated module.

EXAMPLES

```
EXCLUDE PROGA, SUB1, SUB3
EXC ZAP,ABC
```

EXIT

Causes Operating System To Load, And Transfer To Job Control

The EXIT Directive causes the operating system to load and transfer control to Job Control.

SYNTAX

EXI[T]

EXAMPLES

```
EXIT
EXI
```

Incorporates Unreferenced Subprograms in Specified Modules

The INCLUDE Directive causes unreferenced subprograms to be included with a specified module. Only one internal definition of a subprogram needs to be specified to cause that subprogram to be included.

SYNTAX

INC[LUDE] pname,l-name,[,l-name...]

pname - Parameter 1 (required) specifies a program name that indicates the module that includes the subprogram(s).

l-name - Parameter 2 (required) names the internal definitions that indicate the subprograms to be included.

EXAMPLES

```
INCLUDE PROGA,SUB1,SUB3
INC ZAP,ABC
```

INITIALIZE

Removes Associations And Symbols, Then Returns To Initialized State

The INITIALIZE Directive removes all associations made by the XEQUATE Directives and symbols currently in the symbol table, and returns the Link Editors to the initialized state. The Link Editors are automatically initialized when they are loaded by the system.

SYNTAX

INI[TIALIZE]

EXAMPLES

```
INIT
INITIALIZE
INI
```

Sets Up Files For Subprogram Searches

The LIBRARIES Directive sets up a list of library files to be used for subprogram searches before the LIB library is searched. The order of the files in the directive specifies the order in which the libraries are searched. If this directive is not used, the default library files LIL and LIB are used. File LIB is always attached to the end of the list unless the user specifies not to do so by entering the file "=LIB" in the list.

SYNTAX

LIB[RARIES] file [,count],file[,count]]...

file - Parameter 1 (at least one is required) represents the name of the file that contains the library.

coun - Parameter 2 (optional) represents the optional maximum of passes over the library during any library research. If Parameter 2 is not specified, the library is searched repeatedly until no additional modules are loaded during a pass.

Illegal file names in the library list are ignored (M4EDIT).

EXAMPLES

```
LIBRARY A
LIB SSB,1 M,A
LIB SSB
```

MULTIPLE

Edits Modules Or Main Program

The MULTIPLE Directive has a different meaning depending on whether or not the SEGMENT Directive is used (MULTIPLE is the default condition):

- Without SEGMENTATION the Link Editors edit all modules from the bi file from the main program until an end-of-file is reached.

  With SEGMENTATION the Link Editors edit only the main program from the bi file. The entire bi file is considered a library for the rest of the edit.

SYNTAX

MUL[TIPLE]

EXAMPLES

```
MULTIPLE
MUL
```

Writes A Message To The Operator

The NOTE Directive is similar to the ACTION and PAUSE Directives. It causes the Link Editors to write the entire directive on the CO file. Unlike ACTION and PAUSE, NOTE does not cause a HOLD to occur.

SYNTAX

NOT[E] [message]

message = Parameter 1 (optional) represents a message directed to the operator's attention.

EXAMPLE

```
NOTE START OF LINK EDIT.
```

NOXEQUATE (M4EDIT only)

Removes External Associations

The NOXEQUATE Directive removes all associations of External names previously entered by means of an XEQUATE Directive.

SYNTAX

NOX[EQUATE]

EXAMPLES

```
NOX
NOXEQUATE
```

Causes Operands And Instructions To Share MAP

The ONE.MAP Directive causes the module to be formed so that the operands and instructions share the same map addressing space. ONE.MAP is the default condition.

SYNTAX

ONE[MAP]

EXAMPLE

ONE MAP

PAUSE

Writes A Message To The Operator, Then Holds

The PAUSE Directive can be used when the Link Editor's directives are entered from the Job Control procedures. PAUSE directs the operator to intervene at intermediate points during the link-editing process. This directive causes the Link Editor to write the entire directive (PAUSE message) to the CO file, and then to HOLD.

At that point, the Operator Communications task must be activated in order to RESUME execution.

SYNTAX

PAU[SE] [message]

message - Parameter 1 (optional) represents a message directed to the operator's attention.

EXAMPLE

PAUSE ATTENTION: MOUNT MAGTAPE ON MT2

Specifies Only One Object Module To Be Read

The PRIMARY Directive causes the Link Editors to read only one object module from the 81 file in subsequent editing operations. This directive is not affected by segmentation.

SYNTAX

PRI[MARY]

EXAMPLE

PRIMARY

REWIND

Positions Device To Beginning Of Media

The REWIND Directive positions a device to the beginning of the media.

SYNTAX

REW[IND] [file],..

file -Parameter 1 (optional) represents a file name.

EXAMPLES

```
REW B1:80
REWIND,SO,80
```

Continuous/Paging Output To CO

The SCROLL Directive causes continuous output to CO if CI is a terminal.

The NO SCROLL Directive causes the output to CO to be paged if CI is a terminal. Default is NO SCROLL.

SYNTAX

```
SCR[OLL]
NOS[CROLL]
```

EXAMPLES

```
SCR
NOS
```

SEGMENT (M4EDIT only)

Defines Organization Of Segmented Program

The SEGMENT Directive defines the desired organization of a segmented program.

No operation is performed until an EDIT Command is given that causes the object to be generated with the defined structure. Level numbers are used to specify the relative position in the tree. The position in the statement determines the correspondence of host and guest. In most cases, programs and subprograms have a single internal definition whose name corresponds to the program name.

Scratch File A (SCA) is used if the BI file is not random access and the editor is in the MULTIPLE mode.

SYNTAX

SEG[MENT] level,name [,level,name...] .

level - Parameter 1 (at least one is required) is a number in the range 1 to 255 that represents the relative position in the tree structure of the program or subprogram.

name ... - Parameter 2 (at least one is required) represents the names of all the internal definitions in the program or subprogram to be edited at the specified level, separated by commas.

EXAMPLES

```
SEGMENT 1,SUB1,2,SUB2,3,SUB5,SUB6,
SEG 2,SUB2B,3,SUB3
```

Figure 3-1 depicts a <segment-statement> diagram. The general form of the <segment-statement> is the keyword SEGMENT followed by one or more <segments> each of which consists of one or more <level-names> or additional <structures>.

The appearance of a <level> which is less than or equal to the previous <level> signals the end of a branch and establishes that the guests at that <level> cannot have guests. A <name> must start with a letter (A-Z).


<segment statement>

<level-name>
<level-name>
<level-name>

Figure 3-1. Segment Statement Diagram

SYNTAX

segment statement::= SEGMENT segment-list;
segment-list::= segment [,segment]...;
segment::= level-name [,level-name]...;
level-name::= level,name [,name]...;
level::= unsigned integer;
name::= program or subprogram internal definition;

SEMANTICS

The analysis of a statement is from left to right. A tree structure from the first <level> name host must be completed before another host of the same level can be introduced. The <levels> can be introduced only in sequential order to remain in the branch currently being formed. When a <level> is encountered that is less than or equal to the previous <level>, the branch of the previous <level> is terminated and a new branch is formed at the current <level>. If a given program is to be a segment, all internals defined in that program that are referenced by its host must be listed following the level number.

The following statements describe a program organized as follows in Figure 3-2.

```
SEGMENT    1,A1,
SEGMENT         2,A1.1,
SEGMENT         2,BETA,
SEGMENT    1,A2,
SEGMENT         2,A2.1,
SEGMENT                 3,A2.1.1,
SEGMENT                 3,BETA,
SEGMENT         2,A2.2,
SEGMENT    1,A3
```

Figure 3-2 Program Organization Diagram

The use of continuation lines above makes the actual structure of the segmentation more clear.

All the names in the above statement must be defined as internals (in FORTRAN, this is the subprogram name).

The above example statement can also be expressed in the equivalent form:

SEGMENT1 ,A1,2,A1.1,2,BETA,1,A2,2,A2.1,3,A2.1.1,3,BETA,2,A 2.2,1,A3

NOTE: The main program is not mentioned in the SEGMENT Directive.

The <segment-statement> causes a data structure to be built in memory. It controls the final editing of the segmented program. Because the structure is a tree, the processor is free to allocate memory based on the branches in the tree rather than checking maximum sizes.

RESTRICTIONS

There are certain restrictions the user must keep in mind when segmenting a program. A logical maximum of 255 segments is allowed. A practical limit to the number of segments is dictated by the Task/Overlay Cataloger (TOC) that permits 96 segments on a 100 word/sector disc and 124 segments on a 128 word/sector disc. When creating an overlay structure, one must be sure that a given segment is explicitly called only by segments on the level directly above it. In Assembly Language, the call must be an unindexed, direct BLM or BRU for the structure to execute properly (the FORTRAN programmer need not be concerned with this point).

Forms Shared Load Modules

The Shared Load Module (SLM) Directive separates all counters with attributes in the attribute list to form shared load modules. Each occurrence of this directive causes a new module to be generated by the Link Editors. Shared load modules are output on the BO file immediately after the non-shared portions of the program. The order of shared load modules during output are the same as the order of SLM Directives given to the editor. The name parameter must be identical to the name used to catalog each module by the Task/Overlay Catalogers (TOC and TOC32). The Scratch file B (SCB) is used for SLM processing.

The Shared Load Module facility is designed for complete tasks and overlays that must be shared between two or more users. Any program that uses counters and is not cataloged as peculiar relocatable can use this directive to generate Shared Load Modules. If a program is segmented, only the root node can be shared.

CAUTION

SCB is used by TOC as a scratch file in conjunction with this directive. Do not assign BO to SCB in TOC if this directive is used.

SYNTAX

SLM name[,att]...

name - Parameter 1 (required) is the name of the shared module to be created and inserted.

att - Parameter 2 optional, is any relocatable attribute set number.

IMAP and OMAP attribute set numbers can be mixed only for ONEMAP tasks. Multiple attributes must be contiguous in terms of virtual addressing space; shared attributes are not permitted.

EXAMPLES

```
SLM ABC,2,4,5,11
SLM MODULE
```

Cataloging Shared Load Module Programs

All resources needed by the shared program must be cataloged with the non-shared portion of the program. Due to the fact that shared load modules may not have to be loaded by the module loader, the shared module itself should not contain any resources.

NOTE: Shared load modules must not be cataloged peculiar relocatable.

After cataloging the shared program's modules the program can be executed by activating the non-shared portion of the module. The MAX IV module loader loads or inserts each shared portion of the program when it is invoked to load the non-shared portion of the program.

Causes Operands And Instructions To Use Different MAPS

The TWOMAP Directive causes the module to be formed so that different operand and instruction maps are used. ONEMAP is the default condition.

SYNTAX

TWO[MAP]

EXAMPLES

```
TWOMAP
TWO
```

WEOF

Writes End Of File Mark

The WEOF Directive writes a file mark on each logical file name specified.

SYNTAX

WEO[F] file[,file]...

file - Parameter 1 (at least one is required) is a file name.

EXAMPLES

```
WEO BO
WEOF BO,SO,SC
```

Defines Extended Common Blocks

The XCO Directive defines the name of all extended common blocks resolved by the current edit. XCO can be used only once. If XCO is used, an insert global common REX call is generated. Otherwise, extended common blocks are resolved and the user is responsible for inserting the region.

SYNTAX

XCO name

name - Parameter 1 (required) is the name of a global extended common block to be inserted at load time.

EXAMPLES

```
XCO ABC
XCO BIGDATA
```

XEQUATE

Associates Two Parameters In Symbol Table

The XEQUATE Directive associates "name-1" with "name-2" in the symbol table so that External references made to "name-1" are satisfied by internal references of "name-2". If an Internal definition of "name-1" is encountered during the editing of an object module, it is ignored. If "name-1" and "name-2" both appear as External references both are satisfied by Interna. definitions of "name-2"

SYNTAX

XEQ[UATE] [name-1, name-2]...

name-1, name-2... = Parameter 1 (optional) represents symbolic names.

EXAMPLES

```
XEQ ALPHA=BETA
XEQUATE B,C
```

CHAPTER 4
M4EDIT OUTPUT

4.1 LINK EDIT MAP

A link-edit map is output to LO (usually assigned to the line printer) during PASS 2 unless the NOMAP option is designated. An example of a link-edit map follows:

```
AB  MAXIV LINK EDITOR  01/14/84

P/MAIN
   C/COMA   0000
   CTR 2: 0100-0101   CTR 3: 0032-0033  CTR 4: 0200-0201
P/SUBR
   I/SUBR   0102
   CTR 2: 0102-0103   CTR 3: 0034-0036  CTR 4: 0202-0204
HIGH ADDR: 0204   XFER ADDR: 0100
```

The character label, x/, placed in front of a name in the map specifies the nature of that name as follows:

P	indicates PROGRAM NAME
D	indicates DUMMY PROGRAM (segmentation only)
C	indicates COMMON BLOCK
I	indicates INTERNAL NAME
S	indicates a COMMON BLOCK assigned to a shared area (global, private or shared load module).
X	indicates a COMMON BLOCK that resides in extended memory.

Following all the names is a list of start and end addresses for all counters in the given program, CTR being the abbreviation for COUNTER.

HIGH ADDR indicates the highest relocatable address used in the in load module.

XFER ADDR indicates the transfer address for the load module XCOM HI indicates the highest extended common address (assuming a base address of 0) used in the load module.

The example above is for a one-map edit. If the user specifies TWOMAP as one of the directives, the map has a slightly different look. All operand-map addresses are followed by an asterisk (*). A high address is given for both the instruction map and operand map.

CHAPTER 5
SPECIAL CONSIDERATIONS

5.1 PUTTING THE LOCAL: SUBROUTINE ON LB (M4EDIT only)

To use the segmentation feature of the MAX IV Link Editor, the LOCAL: subroutine must be located on a library used in the edit. LB is the logical choice and is used in the following example.

Assumptions:

- LOCAL: source is the second file on rewindable device ABC.
- SCA and SCB are available scratch files.

EXAMPLE

```
$JOB - PUT LOCAL: ON LB
$ASSIGN SI=ABC BO=SCA
$REW SI
$AVF SI 1
$EXECUTE M5A
$WEOF BO
$EXECUTE LIB
ASSIGN BI=LB BO=SCB SI=SCA
REW BO BI SI
ADD 0
COP
ASS BI SCB BO LB
COP
EXIT
$EOJ
```

5.2 TASK/OVERLAY CATALOGER IN CONJUNCTION WITH THE LINK EDITOR (M4EDIT)

If the user includes global shared areas (global common) or private shared areas in the edit using the COMMON Directive, additional references should NOT be made to these areas during catalog time using the Task/Overlay Cataloger (TOC) INSERT Directive.

NOTE: When using the MAX IV/MAX 32 Link Editor, the TOC directive INSERT should never be used.

For more detailed information on TOC, refer to the MAX IV/MAX 32 TASK/OVERLAY CATALOGER, Programmer's Reference Manual.

5.3 BLOCK DATA STRUCTURES

In order to use BLOCK DATA subprograms to initialize common areas, the user should be aware of the following:

- Without segmentation (M4EDIT only).

  In the MULTIPLE mode, all programs on bi are loaded (from the main program until an end-of-file is encountered) including BLOCK DATA (named or unnamed).

  With segmentation

  In the MULTIPLE mode, all unnamed BLOCK DATA subprograms on bi are loaded with the main program (root segment).

- Named BLOCK DATA subprograms

  The FORTRAN compiler allows the explicit naming of a BLOCK DATA subprogram. In order to have the Link Editor load a named BLOCK DATA subprogram, the user must use an INCLUDE Directive with the internal name being the name given to the BLOCK DATA subprogram. This works whether or not segmentation is involved.

APPENDIX A
ERROR MESSAGES

A.1 LINK EDITORS ERROR MESSAGES

Error messages are written on the CO file. If the Control Input device is not a terminal device, the Link Editor enters the HOLD state after the writing of an error message. In such an event, the Operator Communication task must be activated by the operator and a /RESUME Directive must be entered to allow a new directive to be processed. If the Control Input is a terminal device, there is no HOLD so that a new directive may be entered immediately after the writing of an error message.

When input is not from a terminal device, the Dollar option NO$HO can be used to inhibit the Editor from entering the HOLD state. In any case, if an error does occur, the batch GO flag is set to inhibit further processing.

Warning messages do not cause a HOLD or set the GO bit. The following alphabetical list of error messages can be encountered during the edit of binary modules. In addition to the exact message text, a description of the error condition and, where applicable, a suggested recovery procedure is provided.

In addition to the following error messages, LINK32 produces several self-explanatory error messages. Some messages include hexadecimal dumps to be included with the Software Problem Report (SPR) when the error is not obvious.

ABORT (BAD ADD)

Meaning: The executive has returned invalid addressing information on M4EDIT is running.

Recommended Action: Check the attributes of this task.

ABORT (CAN JCN)

Meaning: A can-coded name has been found with an illegal value.

Recommended Action: Check the input file and the libraries.

ABORT (EOF file)

Meaning: A file mark has been unexpectedly encountered on the specified input file.

Recommended Action: Re-create this file.

ABORT (IFC GSR)

Meaning: An illegal function code has been encountered on the specified input file.

Recommended Action: Check the input file and the libraries.

ABORT (RER file)

Meaning: An error has occurred while attempting to read from the specified input file.

Recommended Action: Check the input file assignment and the file itself.

ABORT (R.W. file)

Meaning: An error has occurred while attempting to rewind or write a file mark to the specified file.

Recommended Action: Check the file.

ABORT (STO ALO)

Meaning: The symbol table or output buffers ran the link editor to abort.

Recommended Action: Re-catalog the link editor with a larger space directive.

ABORT (WER file)

Meaning: An error has occurred while attempting to write to the specified output file.

Recommended Action: Check the output file assignment and the file itself.

ATTRIBUTE CHANGE ERROR - COUNTER #nn

Meaning: A counter's access rights attributes (NOR,EXE,REA) have been changed without first page bounding the counter by using a BND 256, or the IMAP/OMAP REL /ABS attributes have been changed.

Recommended Action: Add the BND statement or use unique counter numbers.

CHECKSUM ERROR ON .fn

Meaning: Incorrect data has been read into the editor.

Recommended Action: Check record size of media, reassemble the modules, and re-link edit.

COMMON OVERFLOWS

Meaning: The highest address in common CNAME is greater than the highest address of the shared area SNAME. The message is a warning message only.

Recommended Action: The size of the shared area has not been expanded. If this is unacceptable, the shared area or program must be amended.

COMMON SIZE INCREASED

Meaning: A definition of common CNAME has been met in module MNAME which is larger then the previous from a directive or module. The larger size will be used. This message is a warning only, and may be repeated for the same common if a subsequent module has an even larger definition.

Recommended Action: Check that an increase in common size is acceptable.

DIRECTORY IN ERROR ON lfn

Meaning: A library file contains a directory, but there is an error in the directory.

Recommended Action: Check the format of the library file.

DUPLICATE COMMON DEF'N ERROR - name (M4EDIT only)

Meaning: Two common blocks of the same name have been edited, only one of which was marked as extended.

Recommended Action: Reassemble the modules with matching common types.

DUPLICATE COMMON DEF'N ERROR - name (LNK32 only)

Meaning: Two common blocks of the same name have been defined using the COM directive. The second definition is ignored.

Recommended Action: Reenter the directive using unique names for the common blocks.

DUPLICATE INTERNAL DEF'N ERROR - name

Meaning: Two unique programs have internals defined with identical names.

Recommended Action: Correct internal name conflict and re-link edit.

EDIT HAS WRAPPED AROUND - NOT USABLE (M4EDIT only)

Meaning: The loading address reached the maximum of #FFFF and further loading wrapped around starting location 0. The edit completed and produced a map in order for the user to diagnose the problem. In most cases, the edited module will not load correctly

Recommended Action: Make the program smaller or edit the program in TWOMAP mode.

EXTENDED COMMON HAS WRAPPED AROUND (M4EDIT only)

Meaning: The allocation of extended memory has exceeded the 21-bit size limitation.

Recommended Action: Make the extended common smaller.

FUNCTION ERROR

Meaning: An internal incompatibility between the symbol table and a module input to pass 2.

Recommended Action: Check that the SC scratch file is validly assigned.

ILLEGAL FUNCTION CODE on lfn

Meaning: Normally indicates improper positioning or assigning of the input files.

Recommended Action: Enter an exit directive to return to job control. Check and obtain valid object records then re-link edit.

I/O ERROR ON lfn

Meaning: The MAX IV Basic I/O system has returned an I/O error.

Recommended Action: Re-link edit the modules.

MISSING ROUTINES (M4EDIT only)

Meaning: One or more external definitions have not been satisfied.

Recommended Action: Enter "/RES" to produce an edited module with the unsatisfied external references linked to location zero, or modify the assignments in the library list using the operator communicating facility and continue editing by typing "RES [18." Enter "/RES NO" to terminate the edit and allow entering of a new directive.

PROGRAM NOT FOUND

Meaning: A file mark has been read from the file and the program name specified by "EDIT pname, bi" directive was not found.

Recommended Action: Enter an exit directive "bi" to return to job control. After correcting the error, re-link edit.

SEQUENCE ERROR ON fn

Meaning: The file is not positioned at the beginning of the module or records have been read out of order.

Recommended Action: Check the positioning of the files described in the error message, check the geometry of the file, then re-link edit.

STATEMENT ERROR

Meaning: An unacceptable directive was entered.

Recommended Action: Re-enter a corrected directive (if a terminal device is used) or change the incorrect directive in the job stream.

UNDEFINED EXTERNALS (LNK32 only)
name name

Meaning: One or more externals have not been satisfied. The missing names are displayed under the message.

Recommended Action: Unless LNK32 operates under the NO$HO option, the HOLD state will be entered. At this point the user has a number of options:

- Enter /A to abort LNK32.
- Enter /R to continue the linking with the missing externals linked to location zero.
- Enter /R MAP to continue as above, but unconditionally produce a map.
- Change library file assignments and enter /R LIB to scan the libraries once more.
  Enter /R NO to terminate the edit and allow entry of a new directive.

WRAPAROUND IN CTR #nn PGM name

Meaning: The given counter in a program has wrapped around on itself and cannot be correctly edited.

Recommended Action: The program must be corrected or shortened and re-link edit.

A.2 RUN-TIME ERROR MESSAGE FOR SEGMENTED PROGRAM (M4EDIT only)

If a segmented program is built with M4EDIT, the load-on-call routine (LOCAL:) is added to the program.

While the program is executing, segments are loaded into memory as required. If an error occurs during a segment load, a standard error message is given by the Module Loader.

After a segment load failure, control is passed back to the LOCAL: routine, which causes a program abort of the form:

ABORT (SEG LOD xxx)

to denote a segment load failure.

APPENDIX B
CONVERTING SEGMENTED PROGRAMS FROM EDIT TO M4EDIT

Although the MAX III/IV Link Editor (EDIT) is used on the MAX IV system, it requires that program segmentation is still thought of in MAX III terms.

The MAX IV Link Editor (M4EDIT) offers a more powerful concept of program segmentation. It does not accept all EDIT Directives, and can require the removal of explicit overlay-loading code from the program to be linked. The following sections describe the general cases of programs that link with EDIT and that need to be converted to link with M4EDIT. It is assumed that the reader understands the sections of segmentation and the SEGMENT Directive, described in this manual.

For programs in languages other than FORTRAN or Assembler, any sections on link-editing in the appropriate language Reference Manual must take precedence over information in this document.

B.1 LOAD-ON-CALL PROGRAMS

The MAX III/IV Load-on-Call facility is similar to, but a restricted form of, the MAX IV Segmentation facility. LOCALL cannot be used in EDIT under MAX IV.

When converting a program using this facility from MAX III/IV to MAX IV, the EDIT directives of the form

```
LOCALL A,B,C
```

should be changed to the M4EDIT directives of the form

```
SEGMENT 1,A,1,B,1,C
```

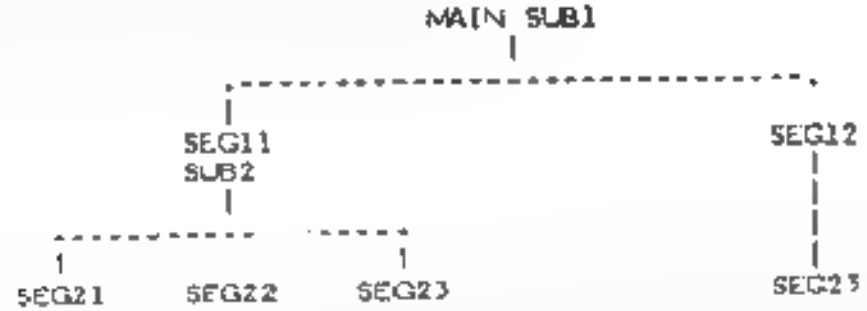
For the maximum number of segments allowed, refer to Chapter 3.

As there is no explicit overlay loading in such a program, there are usually no source changes.

B.2 OVERLAYED PROGRAMS

From a diagram of a MAX III/IV overlay structure, it should be possible to construct a MAX IV segment tree. With segmentation, a segment can be explicitly called only by segments on the level directly above it. This can require additional branches, or moving subprograms up the tree. For example:

If SEG11 calls SEG21, SEG22 and SEG23, and if SEG12 calls SEG23 and SUB1 only we could convert this to



From the segment tree, SEGMENT Directives can be written. Routines can be forced in a segment before they are called by using the INCLUDE Directives.

If the module names above are also the names of the external definitions, the above example might become

```
INCLUDE     SEG11,SUB2
SEGMENT     1,SEG11
SEGMENT     2,SEG21,2,SEG22,2,SEG23
SEGMENT     1,SEG12
SEGMENT     2,SEG23
EDIT
```

These replace the use of the LEVEL, CLOSE and OPEN Directives of EDIT.

All explicit overlay loading needs to be removed from the program, and all overlay running replaced by a direct call. For example if a FORTRAN program contains the lines

```
      DIMENSION N(4)
      DATA N(1),N(2),N(3)/2HSE,2HG2,2H1 /
      ...
      CALL OVERL(N(1))
      CALL OVRUN(N)
```

this should be replaced by

```
      CALL SEG21
```

FORTRAN subprograms described as OVERLAY can alternatively be described as SUBROUTINE when they are linked with M4EDIT.

B.3 COMMON DATA AREAS

GLOBAL COMMON

Because MAX IV has the concept of virtual addressing, the handling of "Global Common" between different tasks is more complex than under MAX II/IV.

If a program is to be converted directly from MAX II/IV to linking by M4EDIT under MAX IV, global common may be inserted by M4EDIT directives.

Suppose named common blocks A and B are contained in a shared area ALPHA, and mapped over by several MAX IV tasks.

The EDIT Directive of the form

```
GCOM A,A-address,A-size
GCOM B,B-address,B-size
```

can be replaced by the M4EDIT Directives

```
ATTRIBUTE ALPHA,GLOBAL
    COMMON/ALPHA,size/A,A-size,B,B-size
```

A PAGESHARE Directive is needed when cataloging the task with TOC. An INSERT Directive to TOC must not be used.

If a global common area is inserted into an EDIT-linked program under MAX IV by either the TOC INSERT Directive or a REX call, the user is strongly recommended to change to the above method, and remove the INSERT Directive and/or REX calls (EXTSHARE and INSGLO in FORTRAN).

As an alternative, directives of the form

```
GCOM A,address,size
```

can be converted to

```
COMMON //A,size,address
```

The user must ensure (e.g. with a BIAS Directive) that this virtual addressing space is not allocated to part of the program being linked, as M4EDIT does not reserve dedicated addressing space for the common when an address parameter is given.

The COMMON Directive forces actual pages of memory to be allocated for it. If the shared area is inserted by REX call, a REX memory deallocate must now precede it.

LOCAL COMMON

M4EDIT first links the root node and then each segment node. A named common block that is not global is contained in the first node that contains a subprogram referencing it. The size is the largest of any of the definitions in the subprograms of node. If a segment lower in the tree contains a definition with a larger size, mis-linking may occur. The directive

COMMON // name,size

forces the larger size for the common, and allocates it in the root node.

If the same common name appears in nodes that are not directly connected to each other, separate data areas are created. If these should be the same area, the common can be moved to the root by the COMMON Directive.

BLOCK DATA STRUCTURE

In order to use FORTRAN BLOCK DATA subprograms to initialize common areas, when linking with M4EDIT, the user should be aware of the following:

- Without segmentation

  In the MULTIPLE mode, all programs on blk is loaded (from the main program until an end-of-file is encountered) including BLOCK DATA (named or unnamed).

- With segmentation

  In the MULTIPLE mode, all unnamed BLOCK DATA subprograms on blk are loaded with the main program (root segment).

  Named BLOCK DATA subprograms

  The FORTRAN compiler allows the explicit naming of a BLOCK DATA subprogram. In order to have the Link Editor load a named BLOCK DATA subprogram, the user must use an INCLUDE Directive with the internal name being the name given to the BLOCK DATA subprogram. This works whether or not segmentation is involved.

B.4 RESTRICTIONS

NUMBER OF SEGMENTS

M4EDIT allows a maximum of 255 segments in a segmented program. The Task/Overlay Cataloger however is limited by the sector size of the disc device it is cataloging onto.

The current restrictions are:

All discs	Maximum 124 segments

CALLING OF SEGMENTS

When creating an overlay structure, be sure that a given segment is explicitly called only by segments on the level directly above it.

Secondly, in Assembly Language, the call must be an unindexed, direct BLM or BRM for the structure to execute properly. This does not affect FORTRAN users.

For programs in other languages, refer to any sections on link-editing in the appropriate Language Reference Manual for any special linking considerations.

(

INDEX

XCD, 3-45
XEQUATE, 3-46

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 3624 FT. LAUDERDALE, FL 33309

POSTAGE WILL BE PAID BY ADDRESSEE

MODULAR COMPUTER SYSTEMS
1650 W. McNAB ROAD
P.O. BOX 6099
FT. LAUDERDALE, FLORIDA 33310

Attention: TECHNICAL PUBLICATIONS, M.S. #85

Fold

MODCOMP

Please comment on the publication's completeness, accuracy, and readability. We also appreciate any general suggestions you may have to improve this publication.

If you found errors in this publication, please specify the page number or include a copy of the page with your remarks.

Your comments will be promptly investigated and appropriate action will be taken. If you require a written answer, please check the box and include your address below.

☐

Comments: _____

Manual Title _____

Manual Order Number _____ Issue Date _____

Name _____ Position _____

Company _____

Address _____

_____ Telephone ( ) _____

MODCOMP

Corporate Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 8099, Ft. Lauderdale, FL 33310; Tel: (305) 974-1380, TWX: 310-372-7837

International Headquarters:
MODULAR COMPUTER SERVICES, Inc., The Business Centre, Molly Millars Lane, Wokingham, Berkshire, RG11 2QQ, UK, Tel: 0734-786908, TLX: 851849149

Latin American Sales Headquarters:
MODULAR COMPUTER SYSTEMS, Inc., 1650 West McNab Road, P.O. Box 8099, Ft. Lauderdale, FL 33310, Tel: (305) 975-6562, TLX: 3727852

Canadian Headquarters:
MODCOMP Canada, Ltd., 400 Matheson Blvd. East, Unit 24, Mississauga, Ontario, Canada L4Z 1N8, Tel: (416) 890-0666, TELEX: 03-861279

SALES & SERVICE LOCATIONS THROUGHOUT THE WORLD

Printed in the U.S.A.